Finding Defective Modules from Highly Unbalanced Datasets

J C Riquelme[1], R Ruiz[2], D Rodríguez[3], and J Moreno[3]

[1] Department of Computer Science
University of Seville
Avd. Reina Mercedes s/n
41012 Sevilla
riquelme@us.es
[2] Universidad Pablo de Olavide
Ctra Utrera Km 1
41013 Sevilla, Spain
robertoruiz@upo.es
[3] Department of Computer Science
University of Alcalá
28805 Alcalá de Henares, Madrid, Spain
daniel.rodriguezg@uah.es

Abstract. Many software engineering datasets are highly unbalanced, i.e., the number of instances of a one class outnumber the number of instances of the other class. In this work, we analyse two balancing techniques with two common classification algorithms using five open public datasets from the PROMISE repository in order to find defective modules. The results show that although balancing techniques may not improve the percentage of correctly classified instances, they do improve the AUC measure, i.e., they classify better those instances that belong to the minority class from the minority class.
Keywords: Balancing techniques, Classification, Software Engineering datasets, defect prediction.

1 Introduction

The number of software engineering repositories containing project management data, source code data, etc. is increasing mainly due to automated data collection tools that allow managers and developers to collect large amounts of information. However, dealing with such large amount of information convey associated problems. For example, when dealing with defect prediction, most datasets are highly unbalanced, i.e., the number of instances of the majority class (non defective modules) outnumber the number of instances of the other class (defective module). In such cases, data mining algorithms do not generate optimal classification models to predict future defective modules. In the data mining literature, different balancing techniques have been proposed to overcome this problem.

In this work, we compare two balancing techniques and two classification algorithms to the problem of finding defective modules. To do so, we have analysed two common classification algorithms and two balancing techniques with five open public datasets from the PROMISE repository[4].

The rest of the paper is organised as follows. Section 2 cover the background about balancing and classification techniques. Section 3 contains the related work. Section 4 discusses the experimental work carried out and the results. Finally, Section 5 concludes the paper and highlights future research work.

2 Background

2.1 Unbalanced Datasets

With unbalanced datasets, the generated models can be suboptimal as most data mining algorithms assume balanced datasets. In such cases, there are two alternatives, either (i) to apply algorithms that are robust to unbalanced datasets or (ii) balance the data using sampling techniques before applying the data mining algorithm. Sampling or balancing techniques can be classified into two groups:

- *Over-sampling*. This group of algorithms aim to balance the class distribution increasing the minority class.
- *Under-sampling*. This group of algorithms tries to balance the class removing instances from the majority classes.

The simplest techniques in each group are Random Over-Sampling (ROS) and Random UnderSampling (RUS). In ROS, instances of the minority class are randomly duplicated. On the contrary, in RUS, instances of the majority class are randomly removed from the dataset.

In order to improve on the performance of random sampling, there are other techniques that apply more sophisticated approaches when adding or removing instances from a dataset. Within the undersampling group, Kubat and Matwin [1] proposed a technique called One-Sided Selection (OSS). Instead of removing instances from the mayority class randomly, OSS attempts to undersample the majority class by removing instances that are considered either noisy or borderline. The selection of noise or borderline instances is carried out applying the Tomlek links [2]. Another undersampling technique is the Neighbourhood Cleaning Rule (NCL) [3] uses Wilson's Edited Nearest Neighbour Rule (ENN) [4] to remove instances from the majority class when two out of three of the nearest neighbors of an instance contradict the class.

As a non-random oversampling method, Chawla *et al.* [5] proposed a method called Synthetic Minority Oversampling Technique (SMOTE). Instead of randomly duplicating instances, SMOTE generates new synthetic instances by extrapolating existing minority instances with random instances obtained from k

[4] `http://promisedata.org/`

nearest neighbors. The technique first finds the k nearest neighbors of the minority class for each minority example (authors used $k = 5$). Then the new samples are generated in the direction of some or all of the nearest neighbors, depending on the amount of oversampling desired.

2.2 Classification Methods

There are many classification methods but to analyze the effectiveness of balancing techniques, here we use two different and well-known types of classifiers, a decision tree classifier (C4.5) and a probabilistic classifier (Naïve Bayes). These two algorithms have also been selected because they represent different approaches to learning.

- The *naïve Bayes* [6] algorithm uses the Bayes theorem to predict the class for each case, assuming that the predictive attributes are independent given a category. A Bayesian classifier assigns a set of attributes $A_1, \ldots, A_n$ to a class C such that $P(C|A_1, \ldots, A_n)$ is maximum, i.e., the probability of the class description value given the attribute instances, is maximal.
- *C4.5* [7]. A decision tree is constructed in a top-down approach. The leaves of the tree correspond to classes, nodes correspond to features, and branches to their associated values. To classify a new instance, one simply examines the features tested at the nodes of the tree and follows the branches corresponding to their observed values in the instance. Upon reaching a leaf, the process terminates, and the class at the leaf is assigned to the instance. C4.5 uses the gain ratio criterion to select the attribute to be at every node of the tree.

3 Related Work

There is a large set of literature related to defect prediction using statistical regression techniques (e.g. Basili *et al.* [8] to data mining. For example, Khoshgoftaar *et al.* [9] which used neural networks for quality prediction. Authors used a dataset from a telecommunications system and compare the neural networks results with with a non-parametric model. Also, Khoshgoftaar *et al.* [**?**] have applied regression trees as classification model to the same problem. Fenton *et al.* [10] propose Bayesian networks as a probabilistic technique to estimate defects among other parameters.

In relation to the problem of unbalanced datasets, sampling techniques are the most widely used. For example, Seiffert *et al.* [11] studied the reduction of noise in the data using a large number of sampling techniques such as *Random undersampling* (RUS), *random oversampling* (ROS), *one-sided selection* (OSS), *cluster-based oversampling* (CBOS), Wilson's editing (WE), SMOTE (SM) and *borderline*-SMOTE (BSM). Authors concluded that (RUS, WE and BSM) are more robust than (ROS and SM) and that the unbalanced level affects the generation of optimum models. For this work, authors used only a public dataset

and generated a number of artificial datasets but extended in another work [**?**] with further methods and datasets from the UCI repository.

In the software engineering domain, Yasutaka *et al.* [12] analyzed four sampling techniques (ROS, SMOTE, RUS, ONESS) to balance a dataset before applying three classification algorithms (lineal discriminant analysis – LDA, Logistic Regression – LR, Neural Networks – NN and Classification Trees – CT) in defect prediction. In this study, sampling improved the classification accuracy of LDA and LR but not for NN and CT. Authors, however, used only one dataset that was not publicly available.

Applying the approach of using robust algorithms to handle unbalanced datasets, Li and Reformat [13] describe a algorithm based on fuzzy logic for this kind of problem.

As the PROMISE repository is publicly available, there are several works applying the same datasets to the ones used here. For example, Menzies *et al.* [14] have used have applied J48 (an implementation of C4.5) and Naïve Bayes to several datasets of the repository to predict defects concluding that are good estimators and suggesting as part of future work a resource bound exploration.

4 Experimental Work

4.1 Datasets

In this paper, we use the CM1, KC1, KC2, and PC1 datasets available in the PROMISE repository [15] to generate models for defect classification. All these datasets were created from projects carried out at NASA and collected under their metrics[5]. Table 1 shows the number of instances (modules) for each dataset with the number of defective, non-defective and their percentage showing that all are highly unbalanced, varying from 7% to 20%. The last attribute is the programming language used to develop those modules.

Table 1. Dataset used in this work.

Dataset	*# of instances*	*Non-defective modules*	*Defective*	*% defective*	*Language*
CM1	498	449	49	9.83	C
JM1	10,885	8,779	2,106	19.35	C
KC1	2,109	1,783	326	15.45	C++
KC2	522	415	107	20.49	C++
PC1	1,109	1,032	77	6.94	C

All datasets contain the same 22 attributes composed of 5 different lines of code measure, 3 McCabe metrics [16], 4 base Halstead measures [17], 8 derived Halstead measures [17], a branch-count, and the last attribute is a binary class

[5] `http://mdp.ivv.nasa.gov/`

with two possible values (false or true, whether the module has or not reported any defects).

McCabe metrics were introduced in 1976 and are based on the count of the number of paths contained in program based on its graph. To find the complexity, the program, module, method of class in an object oriented programming is represented as a graph, and its complexity can be calculated as: $v(g) = e - n + 2$, where e is the number of edges of the graph and n is the number of nodes in the graph. The cyclomatic complexity measures quantity, but McCabe also defined *essential complexity*, `ev(g)`, to measure the quality of the code (avoiding what is known as *spaghetti* code). Structured programming only requires sequences, selection and iteration structures, and the *essential complexity* is calculated in the same manner than the cyclomatic complexity but from a simplified graph where such structures have been removed. The *design complexity* metric, `iv(g)`, is similar, but taking into account calls to other modules.

Another set of metrics is known as Halstead's Software Science also were developed at the end of the 70s. These metrics are based on simple counts of tokens (using compilers's jargon), grouping those into: (i) *operators* such as keywords from programming languages such as `IF THEN`, `READ`, `FOR`; arithmetic operators `+`, `-`, `*`, etc; relational operators ($>$, $\leq$, etc.) and logical operators (`AND`, `EQUAL`, etc.); and (ii) *operands* that include variables, literals and constants. Halstead distinguishes between the number of unique operators and operands and total number of them. Table 2 summarises the metrics collected for all of the datasets. Table 2 summarizes the metrics collected from the datasets.

4.2 Execution and Results of the Experiments

The experiments were conducted using WEKA [18]. As sampling methods in this work, we have use *resample* implementation of WEKA which replicates instances randomly of and our implementation of SMOTE[6]. The classification methods were already implemented in WEKA. The results reported in this section were obtained 10 cross-validation, i.e., data is divided into 10 bins using the 90% of the instances for training and then, 10% for testing. The procedure is repeated 10 times so all data is used for both training and testing and the reported results are the average of those 10 runs.

In the case of balanced datasets, it is reasonable to use *accuracy* to measure performance. However, with unbalanced datasets, models may not consistently produce a useful balance between false positive rates and false negative rates. The ROC (Receiver Operating Characteristic) graphs are used to analyse the relationship between *true positive rate* and *true negative rate*. However, the *Area Under the ROC Curve* (AUC) is generally used to provide a single value of the performance. *AUC* is a useful metric for classifier performance as it is independent of the decision criterion selected and prior probabilities.

These measures have been used to compare the accuracy of the classifiers with and without balancing techniques. The results of the experiment can be

[6] The implementation of SMOTE is available at `http://www.ieru.org/wiki`

Table 2. Attribute Definition Summary

	Metric	*Definition*
McCabe	loc	McCabe's Lines of code
	v(g)	Cyclomatic complexity
	ev(g)	Essential complexity
	iv(g)	Design complexity
Halstead base	uniq_Op	Unique operators, n_1
	uniq_Opnd	Unique operands, n_2
	total_Op	Total operators, N_1
	total_Opnd	Total operands N_2
Halstead Derived	n	Vocabulary, $n = n_1 + n_2$
	L	Program length, $N = N_1 + N_2$
		Estimated Length: $N' = n_1 \cdot log_2 n_1 + n_2 \cdot log_2 n_2$
	V	Volume, $V = N \cdot log_2(n)$
	d	Difficulty $D = 1/L$
	i	Intelligence
	e	Effort $e = V/L$
		estimated $e = n_1 N_2 N log_2 n / 2 n_2$ (elementary mental discriminations)
	b	Error Estimate
	t	Time estimator $T = E/18$ seconds
	lOCode	Line count of statement
	lOComment	Count of lines of comments
	lOBlank	Count of blank lines
	lOCodeAndComment	Count of lines of code and comments
Branch	branchCount	No. branches of the flow graph
Class	false, true	Whether the module has reported defects

observed in tables 3 and 4. The percentage of correctly classified instances did not increase, but the AUC improves when using sampling techniques, especially with SMOTE.

Table 3. Percentage of Correctly Classified Instances

	CM1	*JM1*	*KC1*	*KC2*	*PC1*
J48	87.95	79.50	84.54	81.42	93.33
Naïve Bayes	85.34	80.42	82.36	83.52	89.18
Resample & J48	91.57	86.84	89.81	90.80	94.86
Resample & Naïve Bayes	83.94	79.40	81.32	85.82	85.03
SMOTE & J48	84.28	77.27	82.75	80.13	88.34
SMOTE & Naïve Bayes	78.43	70.93	76.18	75.68	83.62

Table 4. AUC Values

	CM1	*JM1*	*KC1*	*KC2*	*PC1*
J48	0.56	0.65	0.69	0.70	0.67
Naïve Bayes	0.66	0.68	0.79	0.83	0.65
Resample & J48	0.80	0.80	0.82	0.84	0.80
Resample & Naïve Bayes	0.72	0.69	0.78	0.87	0.67
SMOTE & J48	0.77	0.75	0.79	0.77	0.77
SMOTE & Naïve Bayes	0.72	0.68	0.79	0.83	0.61

5 Conclusions and Future Work

In this paper, we analysed two balancing techniques and two classification algorithms in order to find defective modules in five publicly available software engineering datasets. The results show that although balancing technique do not improve the percentage of correctly classified instances, they do however improve the AUC measure, i.e., they classify better those instances from the minority class which are the ones of interest (in this case, to find defective modules). Also, SMOTE seems to improve the AUC measure over the resampling method.

Future work will consist of implementing further balancing algorithms and how to combine balancing techniques and feature selection of attributes when datasets are highly unbalanced.

Acknowledgements

This research was supported by Spanish Research Agency (TIN2007–68084–C02–00) and project CCG07-UAH-TIC-1588 (jointly supported by the Univer-

sity of Alcalá and the autonomic community of Madrid). Also to the Universities of Seville, Pablo de Olavide and Alcalá.

References

1. Kubat, M., Matwin, S.: Addressing the curse of imbalanced training sets: one-sided selection. In: Proc. 14th International Conference on Machine Learning, Morgan Kaufmann (1997) 179–186
2. Tomek, I.: Two modifications of cnn. IEEE Transactions on Systems, Man and Cybernetics **6**(11) (November 1976) 769–772
3. Laurikkala, J.: Improving identification of difficult small classes by balancing class distribution. Technical Report Technical Report A-2001-2, University of Tampere (2001)
4. Wilson, D.L.: Asymptotic properties of nearest neighbor rules using edited data. Systems, Man and Cybernetics, IEEE Transactions on **2**(3) (July 1972) 408–421
5. Chawla, N.V., Kevin W. Bowyer, Lawrence O. Hall, W.P.K.: Smote: Synthetic minority over-sampling technique
6. Mitchell, T.: Machine Learning. McGraw Hill (1997)
7. Quinlan, J.R.: C4.5: Programs for machine learning. Morgan Kaufmann, San Mateo, California (1993)
8. Basili, V., Briand, L., Melo, W.: A validation of object-oriented design metrics as quality indicators. IEEE Transactions on Software Engineering **22**(10) (1996) 751–761
9. Khoshgoftaar, T., Allen, E., Hudepohl, J., Aud, S.: Application of neural networks to software quality modeling of a very large telecommunications system. IEEE Transactions on Neural Networks **8**(4) (1997) 902–909
10. Fenton, N., M.Neil, Krause, P.: Software measurement: uncertainty and causal modeling. IEEE Software **19** (2002)
11. Seiffert, C., Van Hulse, T.M.K.J., Folleco, A.: An empirical study of the classification performance of learners on imbalanced and noisy software quality data. IEEE International Conference on Information Reuse and Integration (IRI 2007). (13–15 Aug. 2007) 651–658
12. Kamei, Y., Monden, A., Matsumoto, S., ichi Matsumoto, T.K.K.: The effects of over and under sampling on fault-prone module detection. In: Empirical Software Engineering and Measurement (ESEM 2007). (September 2007) 196–204
13. Li, Z., Reformat, M.: A practical method for the software fault-prediction. IEEE International Conference Information Reuse and Integration (IRI 2007) (13-15 Aug. 200) 659–666
14. Menzies, T., Greenwald, J., Frank, A.: Data mining static code attributes to learn defect predictors. IEEE Transactions on Software Engineering **33**(1) (2007) 2–13
15. Boetticher, G., Menzies, T., Ostrand, T.: Promise repository of empirical software engineering data (`http://promisedata.org/`). Technical report, West Virginia University (2008)
16. McCabe, T.J.: A complexity measure. IEEE Transactions on Software Engineering **2**(4) (December 1976) 308–320
17. Halstead, M.: Elements of Software Science. Elsevier (1977)
18. Witten, I., Frank, E.: Data Mining: Practical machine learning tools and techniques. 2 edn. Morgan Kaufmann, San Francisco (2005)